Robot Actions Planning and Execution Control for Autonomous Exploration Rovers

Matthieu Gallien **Félix Ingrand** **Solange Lemai**
LAAS-CNRS*
7, Avenue du Colonel Roche
31077 Toulouse Cedex 4, France

Abstract

To achieve the ever increasing demand for science returns, extraterrestrial exploration rovers require more autonomy to successfully perform their missions. Indeed, the communication delays are such that teleoperation is unrealistic. Although the current rovers (such as MER) demonstrate a limited navigation autonomy, and mostly rely on ground mission planning, the next generation (e.g. NASA Mars Science Laboratory and ESA Exomars) aims at "beyond the field of view" autonomous navigation. Other exploration missions which cannot rely on human teleprogramming, will even require activity planning, repair and replanning to be made onboard.

In this paper, we propose and give experimental results of an original approach for temporal planning and execution control, including plan repair and replanning, fully integrated onboard a robot performing rover exploration like missions. Our claim is twofold. First these planning/plan repair methods and techniques are now mature enough to be considered to solve real world problems. Second they can be integrated in existing architectures and used onboard a fully operational robot, with currently available hardware.

Introduction

Extraterrestrial exploration rovers have an increasing need for high level autonomy. If one compares the navigation capabilities of Sejourner and MER, one can already see that some modest, yet real, navigation autonomy has been introduced. Moreover, higher science return, and the communication latency of deep space mission[1] are pushing to get some of the traditionally high level activities planning performed on board. For example in the MER mission, an automated planning system (MapGen (Ai-Chang *et al.* 2003)) was used on the ground to produce the daily activities for Spirit and Opportunity. The operational results of MapGen are quite encouraging, as it allowed a 25% increase in science returns compared to a human generated plan (Rajan 2004). As of today, the ESA Exomars project (part of Aurora) aims at having the rover navigating over its "field of view", in one day, with navigation decisions taken on board. NASA MSL will also push the autonomy cursor further than for MER. Last, the "Human on Mars" goal will require the deployment of a large number of autonomous systems to "prepare" and study the planet before a human can set foot on it. The "future" of exploration rovers and probes clearly lies in an increased autonomy addressing the problems of action planning, and plan execution control.

Meanwhile, automated actions planning has made some progress since the early days of Shakey and STRIPS. There are now planners able to take into account time, resources, constraints and to solve real world problems. Still, planning is only one aspect of the problem. Plans, even flexible or contingent one, are bound to fail. Plan repair and replanning are thus needed to ensure that the system is able to recover from unexpected plan execution failure.

In this paper we present IxTeT, a temporal planner which includes an execution controller, as well as some plan repair and replanning capabilities. The resulting system has been integrated in the LAAS architecture(Alami *et al.* 1998) and implemented onboard Dala, our iRobot ATRV Robot. Such a planner is in charge of producing plans composed of actions such as move, science activities (moving and operating instruments), communication with earth and an orbiter or a lander, while managing resources (power, memory, etc) and temporal constraints (communication visibility windows, rendezvous, etc).

Still, the execution of action as simple navigation task such as a move in an unknown environment implies complex processes (Lacroix *et al.* 2003; Goldberg, Maimone, & Matthies 2002): localization, map building, motion generation, etc. The LAAS architecture (Alami *et al.* 1998) and its associated tools provide a support in order to design and integrate such a complete autonomous system.

Fig. 1 presents the architecture implemented for the experiment on Dala. The *functional level* includes all

*List of authors in alphabetical order. Part of this work has been funded by a grant from the ESF (European Social Fund), and is partially supported by CNES and Astrium

[1]Unlike most navigations on the ground, a comet landing phase can hardly be suspended.

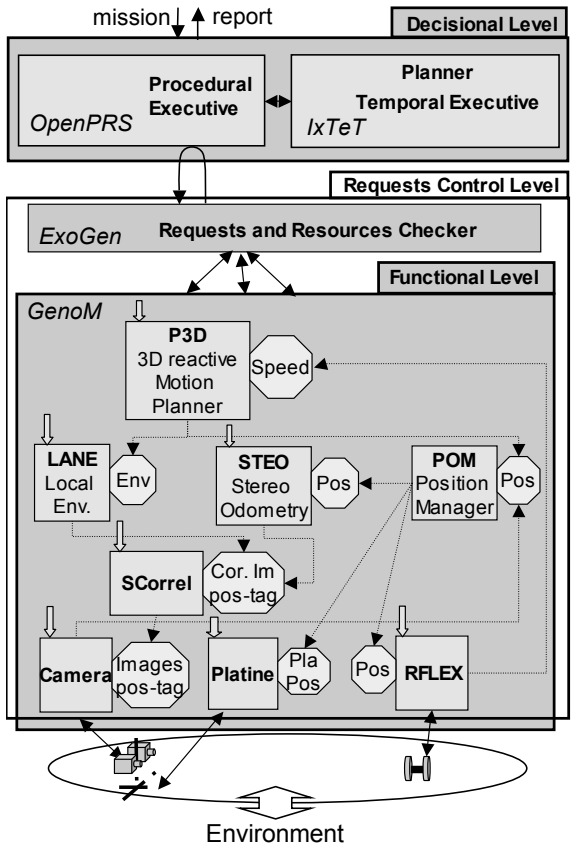

Figure 1: The LAAS architecture on Dala, an iRobot ATRV.

the basic built-in robot action and perception capabilities, encapsulated into controllable communicating modules. These modules are activated by requests, send reports upon completion and export data. Tor example, the POM module computes the best position estimate from standard (RFLEX) and visual (STEO) odometry, while the wheels are controlled by RFLEX according to the reference velocity produced by the reactive motion planner (P3D). The *requests control level* filters the requests according to the current state of the system and a formal model of allowed and forbidden states (see (Py & Ingrand 2004)).

IxTeT has been integrated in the *decisional level* and interacts with the user and the functional level through a procedural executive (OpenPRS). First, IxTeT produces a plan to achieve a set of goals provided by the user. The plan execution is controlled by both procedural and temporal executives as follows. The temporal executive decides when to start or stop an action in the plan and handles plan adaptations. OpenPRS expands and refines the action into commands to the functional level, monitors its execution and can recover from specific failures. It finally reports to IxTeT upon the action completion.

The paper is organized as follows. The first section presents the core planner used as well as the 3DC+ algorithm. The following section focuses on the execution control part of the system as well as the repair and replanning mechanism. Then we present the experimentation (rover exploration planning), and the result of the integration of IxTeT on board the Dala robot. Last, we compare this work with similar works and present conclusions and possible prospectives.

The planner

The planner in IxTeT is a lifted POCL temporal planner based on CSPs (Laborie & Ghallab 1995). Its temporal representation describes the world as a set of *attributes*: logical attributes (e.g. robot_position($?r$)), which are multi-valued functions of time, and resource attributes (e.g. battery_level()) for which one can specify borrowings, consumptions or productions. We note $LgcA$ and $RscA$, respectively the sets of logical and resource attributes. $LgcA_g$ and $RscA_g$ designate the sets of all possible instantiations of these attributes.

The evolution of a logical attribute value is represented through the proposition *hold*, which asserts the persistence of a value over a time interval, and the proposition *event*, which states an instantaneous change of value. The propositions *use*, *consume* and *produce* respectively specify over an interval the borrowing, the consumption or the production at a given instant of a resource quantity.

```
task MOVE(?initL,?endL)(st,et){
  ?initL,?endL in LOCATIONS;
  event(ROBOT_POS():(?initL,IDLE_POS),st);
  hold(ROBOT_POS():IDLE_POS,(st,et));
  event(ROBOT_POS():(IDLE_POS,?endL),et);
  event(ROBOT_STATUS():(STILL,MOVING),st);
  hold(ROBOT_STATUS():MOVING,(st,et));
  event(ROBOT_STATUS():(MOVING,STILL),et);
  hold(PTU_POS():FORWARD,(st,et));
  variable ?di,?du,?dist;
  variable ?duration;
  distance(?initL,?endL,?di);
  distance_uncertainty(?du);
  ?dist = ?di * ?du;
  speed(?s);
  ?dist = ?s * ?duration;
  contingent ?duration = et − st;
}latePreemptive
```

Figure 2: Example of move action model.

```
task MOVE_PTU(?initL,?endL)(st,et){
  timepoint end_heat;
  ?initL,?endL in PTU_POSITIONS;
  hold(ROBOT_STATUS():STILL,(end_heat, st));
  event(PTU_STATUS():(COLD, HEAT),st);
  hold(PTU_STATUS():HEAT,(st,end_heat));
  event(PTU_STATUS():(HEAT,MOVING),end_heat);
  hold(PTU_STATUS():MOVING,(end_heat,et));
  event(PTU_STATUS():(MOVING,COLD),et);
  hold(PTU_INIT():TRUE,(st,et));
  hold(PTU_POS():?initL,(st,end_heat));
  event(PTU_POS():(?initL,PTU_POS_IDLE),end_heat);
  hold(PTU_POS():PTU_POS_IDLE,(end_heat,et));
  event(PTU_POS():(PTU_POS_IDLE,?endL),et);
  (end_heat − st) in [10,12];
  contingent (et − st) in [16,20];
}latePreemptive
```

Figure 3: Example of move_ptu action model.

As shown on Fig. 2, an action (also called *task*) consists of a set of *events* describing the change of the world induced by the action, a set of *hold* propositions expressing required conditions or the protection of some fact between two events, a set of resource usages, and a set of constraints on the timepoints and variables of the action. Note the *conteingent* keyword used to express that this duration should not be modified by the planner.

A plan relies on two CSP managers. A Simple Tem-

poral Network (STN) handles the timepoints and their binary constraints (ordering, duration, etc.). The other CSP manages atemporal symbolic and numeric variables and their constraints (binding, domain restriction, sum, etc.). Mixed constraints between temporal and atemporal variables can also be expressed (Trinquart & Ghallab 2001) (e.g. the relation between the distance, speed and duration of a move $?dist = ?speed * (et - st)$). These CSP managers compute for each variable a minimal domain which reflects only the necessary constraints in the plan. Thus the plan is least committed and as much as possible flexibility is left for execution.

The plan search explores a tree $\mathcal{T}$ in the partial plan space. In a POCL framework, a partial plan is generally defined as a 4-tuple (A, C, L, F), where A is a set of partially instantiated actions, C is a set of constraints on the temporal and atemporal variables of actions in A, L is a set of causal links[2] and F is a set of flaws. A partial plan stands for a family of plans. It is considered to be a valid solution if all its possible instances are coherent, that is F is empty.

The root node of $\mathcal{T}$ consists of: the initial state (initial values of all instantiated attributes), expected availability profiles of resources, goals to be achieved (desired values for specific instantiated attributes) and a set of constraints between these elements. The branches of $\mathcal{T}$ correspond to resolvers (new actions or constraints) inserted into the partial plan in order to solve one of its flaws. Three kinds of flaws are considered:
– *Open conditions* are events or assertions that have not yet been established. Resolvers consist in finding an establishing event (in the plan or a new action) and adding a causal link that protects the attribute value between the establishing event and the open condition.
– *Threats* correspond to pairs of *event* and *hold* which values are potentially in conflict. Such conflicts are solved by adding temporal or binding constraints.
– *Resource conflicts* are detected as over-consuming sets of potentially overlapping propositions. Resolvers include insertion of resource production action, etc
Thus, a planning step consists in detecting flaws in the current partial plan, selecting one, choosing a resolver in its associated list of potential resolvers and inserting it into the partial plan. This planning step is repeated until a solution plan is found. When a dead end is reached (flaws remain but no resolver are available), the search backtracks on a previous choice. The algorithm is complete and the flaw and resolver choices are guided by diverse heuristics discussed in (Laborie & Ghallab 1995). Note that the search is stopped as soon as a valid plan is found.

The advantages of the CSP-based functional approach are numerous in the context of plan execution.

[2]A causal link $a_i \xrightarrow{p} a_j$ denotes a commitment by the planner that a proposition p of action a_j is established by an effect of action a_i. The precedence constraint $a_i \prec a_j$ and binding constraints for variables of a_i and a_j appearing in p are in C.

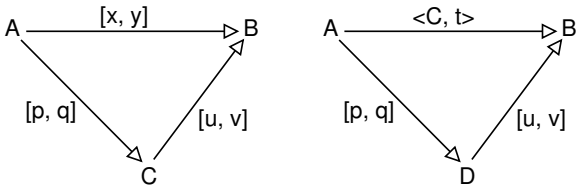

Figure 4: Two Network Examples

Besides the expressiveness of the representation (handling of time and resources), the flexibility of plans (partially ordered and partially instantiated, with minimal constraints) is well-adapted to their execution in an uncertain and dynamic environment. Plans are actually constrained at execution time. Finally, the planner, performing a search in the plan space, can be adapted to incremental planning and plan repair.

3DC+ algorithm

Nevertheles, there are still open problems such as how to handle the controllability issue. Regular propagation in STN, and by extension in the atemporal CSP, may shrink a temporal interval which may not be "controllable" by the planner. As a result, the execution may fail, not because the action model is wrong, but because the planner took some "freedom" with respect to what it is allow to control.

The 3DC+ algorithm was first introduced by (Vidal, Morris, & Muscettola 2001). Fig. 5 presents the general algorithm illustrated on the two examples on fig. 4.

Five various cases must be distinguished. If one considers the network on the left (fig. 4), with a contingent link AB:

Precede case This is the case where $u \geq 0$. In this case we must tighten AB to $[y - v, x - u]$.

Unordered case This is the case where $u < 0$ and $v \geq 0$. In this case and if $x < y - v$, we must add a ternary constraint, called a wait, on AB and of value $< C, y - v >$. It means that we must wait $y - v$ after the instantiation of A to instantiate B. We must also instantiate B at a time consistent with the constraints and after the observation of C.

If one now considers the network on the right (fig. 4):

Regression of wait Suppose a link AC has a wait $< C, t >$

- If a link DB (including AB itself) with an upper bound of q exists, then we must add a wait $< C, t - q >$ on AD.
- If a contingent link DB with $B \neq C$ and with p as lower bound exists, then we must add a wait $< C, t - p >$ on AD.

General reduction If a link AB has a wait $< C, t >$ and the lower bound of the contingent link that ends on C is l with $l < t$, then we must add a lower bound of l on AB.

Unconditional wait If a link AB has a wait $< C, t >$ and the lower bound of the contingent link that ends on C is l with $l > t$, then we must add a lower bound of t on AB and suppress the wait which is useless.

1. Compute the minimal STN. If it is not pseudo-controllable return false.
2. Select any triangle such that v (fig. 4) is non-negative. Introduce any tightenings required by the Precede case and any waits required by the Unordered case.
3. Do all possible regressions of waits, while converting unconditional waits to lower bounds. Also introduce lower bounds as provided by the general reduction.
4. If steps 2 and 3 do not produce any more tightenings, then return true, otherwise return to 1.

Figure 5: 3DC+ Algorithm

We have implemented the 3DC+ algorithm presented above in IxTeT, and we are thus able to produce plans which are dynamically controllable with waits. The resulting plans may not be as "efficient" as one produced without 3DC+, but, as we will see in the example section, it is more robust and still more efficient than a plan where all non controllable actions have been maximized.

Temporal executive, plan execution, repair and replanning

The temporal executive controls the temporal network of the plan produced by IxTeT by deciding the execution order of actions execution and by mapping the timepoints at their execution time. The execution of an action a with grounded parameters p_a, starting timepoint st^a, ending timepoint et^a, and identifier i_a is started by sending the command to the procedural executive. If the action is *non preemptive*, et^a is not controllable, and IxTeT just monitors if a is completed in due time. Otherwise et^a is controllable: if the action does not terminate by itself, it is stopped as soon (resp. as late) as possible if a is *early (resp. late) preemptive*.

IxTeT integrates in the plan the reports sent by the controlled system upon each action completion. A report returns the ending status of the action (*nominal*, *interrupted* or *failed*) and a partial description of the system state. If nominal, it just contains the final levels of the resources, if any, used by the action. Otherwise, it also contains the final values of the other state variables relevant to the action.

Besides completion reports, IxTeT also reacts to user requests to insert a new goal and sudden alterations of a resource capacity.

In any case, while execution is taking place, various events can forbid further execution of the plan:
– *temporal failures* The STN constrains each timepoint t to occur inside a time interval $[t_{lb}, t_{ub}]$. Thus two types of failure lead to an inconsistent plan: the corresponding event (typically, the end of an action) happens *too early* or *too late* (time-out).
– *action failure* The system returns a non nominal report.
– *resource level adjustment* If an action has consumed more or produced less than expected, the plan may contain future resource contentions.

When these occur, IxTeT starts and controls the processes of plan adaptation. To take advantage of the temporal flexibility of the plan, the dynamic replanning strategy has two steps. A first attempt is to repair the plan while executing its valid part in parallel. If this fails or if a timepoint times out, the execution is aborted and IxTeT completely replans from scratch.

Interleaving partial order planning and execution may insert flaws in the plan. We formally define under which conditions such a partial plan remains executable.

Definitions

We extend the previous definition of a partial plan to the definition of P_t: a **partial plan partially executed up to time t**.

Definition 1 $P_t = (RA_t, FA_t, S_t, G_t, C_t, L_t, F_t)$.

RA_t is the set of currently *running actions* ($a \in RA_t$ if $st^a_{ub} < t$ and $et^a_{ub} > t$), FA_t is the set of *future actions* ($a \in FA_t$ if $st^a_{ub} \geq t$). S_t represents the *state of the world* at time t. It is composed of 2 sets: $LgcS_t$ contains the last value of each attribute $la \in LgcA_g$, [3] $RscL_t$ contains the level at time t of each resource $r \in RscA_g$. G_t is the set of *goals* not yet completely achieved at time t (and eventually not established) [4]. C_t is the set of constraints on the variables appearing in FA_t, RA_t, S_t and G_t. L_t is the set of causal links supporting future actions. F_t is the set of flaws present in the partial plan at time t.

The level of a resource at a certain time in the future cannot be computed, since it depends on the partial order of actions using this resource. But at time t the past part of the plan is completely instantiated and linearized. Two cases have to be considered: if no running action modifies r, the exact level can be computed; if at least one action in RA_t requires the resource, only an estimate is available. We refer the reader to (Lemai & Ingrand 2004) for the details on how these evaluations are computed.

A timepoint in the temporal network may correspond to a goal timepoint or to an action starting or ending timepoint.

Definition 2 (executable timepoint) *A timepoint T is executable at time t if all timepoints T^p that must directly precede it in the temporal network have already been executed ($T^p_{lb} = T^p_{ub} < t$), if all positive waits on*

[3] In IxTeT, $LgcS_t$ contains the last executed *event* for each la.

[4] In IxTeT, a goal is represented by a grounded proposition $hold(GoalAtt(g){:}GoalValue, (st^g, et^g))$. G_t contains goals such that $et^g_{ub} \geq t$.

links with positive upper bound and which ends on T are enabled and if $t \in [T_{lb}, T_{ub}]$.

A goal is instantaneously achieved or persistent (achieve and maintain a property between st^g and et^g).

Definition 3 (achievable goal) *A goal g is achievable at time t if* st^g *is executable and if* $g \notin F_t$.

Let A_t^f be the set of actions that are involved in F_t. [5]

Definition 4 (executable action) *A future action a is executable at time t if its start timepoint is executable and if* $a \notin A_t^f$.

Definition 5 (executable plan) *A partial plan* P_t *is executable at time t if the constraint networks are consistent and if* $RA_t \cap A_t^f = \emptyset$.

Execution cycle

As previously explained, the system, when bootstrapped, produces a first plan (let us call it *ExecutingPlan*), and will only start execution afterward. The executive manages the messages received, the actions timeout, and the timepoints execution. Integrating messages in *ExecutingPlan* may partially invalidate it. If *ExecutingPlan* contains new flaws, a *plan repair* consists in keeping the structure of the plan (the ordering of actions) and taking advantage of the flexibility to try and find a solution plan. The user defines the maximum time allowed for plan repair (μ). If plan repair takes more than μ, it is suspended to allow reactivity to events and concurrent execution of the valid part of the plan.

Yet, to distribute planning on several cycles raises two problems:

Which plan does the concurrent execution rely on, especially if no solution has been found? This plan has to be *executable*. At each planning step, the node is labeled if the current partial plan is *executable*. When μ has elapsed, the last labeled partial plan becomes *ExecutingPlan*.

Which plan and which search tree the planning process rely on in the next cycle? If no change has been made meanwhile (no timepoint execution, no message reception), the search tree can be kept as is and further developed during the next *plan repair* part. However, if the plan has been modified, a new search tree whose root node is the new *ExecutingPlan* is used, and the planning decisions made in previous cycles are final.

The following subsections further detail the different phases of the executive loop. Basically, all modifications made to *ExecutingPlan* have to guarantee that an *executable* plan is available after each phase of the cycle. If this condition does not hold, the cycle is stopped and a complete replanning is mandatory. During a cycle without plan repair, *ExecutingPlan* remains a solution plan.

[5] The determination of A_t^f is straightforward in the case of open conditions and resource conflicts. In a threat case, an action a_k has effects in contradiction with the establishment of proposition p by the causal link $a_i \xrightarrow{p} a_j$ and $(a_i \prec a_k \prec a_j)$ is consistent. A_t^f contains a_k and a_j.

Message integration

A message can be a report upon action completion; a new goal request or a notification of a capacity alteration (we do not detail the two last ones, and refer the reader to (Lemai 2004) for a complete explanation on these).

A report is associated with the ending timepoint et^a of the corresponding action a. If the message is received inside the bounds $[et^a_{lb}, et^a_{ub}]$, et^a is set to the current time t (equivalent to posting the constraint $(et^a - origin) = t$ in the STN). Otherwise, two situations arise. If there is no flexibility left in the plan, it is not executable anymore. Else, a new end timepoint, set to t and constrained to occur before the executable timepoints, is created and the failed one is relaxed. The network is then recomputed. In IxTeT, such an operation keeps the network consistent, since the only constraint that can be specified between two actions a and a' is a precedence constraint which upper bound is flexible: $(st^{a'} - et^a)$ in $]0, +\infty[$. If the report contains information about the state, S_t is updated in the following way:

Resource level - For each resource r, the report returns the current "real" level l_r. l_r is compared to the forecasted evaluation (see (Lemai & Ingrand 2004)) which are properly updated accordingly. Plan repair is requested in case of over-consumption and in case of over-production of a reservoir resource (which may then overfill).

State variables - $LgcS_t$ contains the last value for each instantiated logical attribute. If the report is nominal, $LgcS_t$ is updated with the effects of a expected in the plan. Otherwise, it is updated with the values returned in the report. A value is not inserted if it leads to a non executable plan (that is it threatens some proposition of a running action a_r). In that case and if a_r is preemptive, its interruption is requested. Else, the value is inserted and causal links which contradict it are broken. This update leads to an executable plan with open conditions on which plan repair can be processed.

After message integration, the plan may contain flaws (open conditions and/or resource conflicts) on a set of grounded attributes Att^f, possibly repaired thanks to the insertion of new actions. Let us consider Att^i the set of the attributes appearing in the potentially inserted actions. Additional causal links, protecting propositions in the plan on attributes in Att^i, have to be broken to allow the insertion of these actions in the current plan structure.

The determination of Att^i is based on information given by an abstraction hierarchy verifying the Ordered Monotonicity Property (Knoblock 1994; Garcia & Laborie 1995) and generated offline from the model de-

scription. Notably, this hierarchy points out the primary effects of an operator, which justify its insertion to solve a flaw. Let us call *main attributes* of an action the attributes appearing in its primary effects. Att^i, initialized with Att^f, is computed by searching the action operators for which at least one attribute att_m in Att^i is a main attribute. This operator is partially grounded (by binding its corresponding parameter with att_m) and the (eventually grounded) attributes appearing in the operator and not yet taken into account are added to Att^i. The algorithm proceeds recursively until a fixed point is reached.

Finally, the partial plan is executable and the sets of actions that are independent from the failures remain executable.

Plan repair

The plan repair is similar to the IxTeT search process in the plan space. The root of the search tree $\mathcal{T}$ is *ExecutingPlan*, partially invalidated. Planning is distributed, if necessary, on several cycles and each time a new timepoint is inserted, it is constrained to occur after the end of the current cycle. Planning during one cycle is done one step at a time until it results into a dead-end (there is no solution), or a solution is found or a deadline is reached. This deadline corresponds to the user defined time (μ) allocated to the plan repair part of the cycle time.

Some aggregation mechanisms allow a reduction of the search space. In IxTeT, the establishing events are looked for in $LgcS_t$ and executed resource propositions are aggregated in one proposition.

This plan repair process is not guaranteed to find a valid plan, yet it can avoid aborting execution and completely replanning at each failure. By invalidating only a part of the plan, the amount of decisions is rather limited and a repaired plan may be found in a few cycles. Plan repair is especially efficient and useful for temporally flexible plans and plans with some parallelism. This mechanism is also efficient to compensate for inadequate models of actions. Consider a move(L_1,L_2) action, which is defined as a late preemptive action in the IxTeT model. If the robot takes longer than expected in the model (e.g. due to unexpected obstacle avoidance), the action is interrupted. The controlled system returns the intermediate location L_i and, if some temporal flexibility remains, a new move(L_i,L_2) is immediately inserted and launched. This example is representative of the failures that frequently break plan execution.

Action

Each timepoint is associated to an *execution time* t_{exec}. If T is a start or goal timepoint, or an end timepoint of an early preemptive action, $t_{exec} = T_{lb}$. If T is an end timepoint of a late preemptive action, $t_{exec} = T_{ub} - ts$. If T is an end timepoint of a non preemptive action, $t_{exec} = T_{ub}$. The executive determines the set of timepoints to execute during the current cycle (*ExecTPs*): these timepoints are executable and their execution time happens before the end of the cycle. *ExecTPs* is updated after each timepoint execution to take into account newly executable timepoints. The detail of a timepoint execution depends on its type and timeouts are raised when reports have not been received in time.

Complete replanning

Let us call $P_{t_s} = (\emptyset, FA_{t_s}, S_{t_s}, G_{t_s}, C_{t_s}, L_{t_s}, F_{t_s})$ the plan obtained once execution is stopped. An initial plan is extracted from P_{t_s} as:
$P_{t_i} = (\emptyset, \emptyset, S_{t_i}, G_{t_i}, C_{t_i}, \emptyset, F_{t_i})$, with
$S_{t_i} = S_{t_s}$, $G_{t_i} = \{g \in G_{t_s} / \textit{temporal constraints on } g \textit{ are coherent with current time}\}$, $C_{t_i} = \{c \in C_{t_s} / c \textit{ is a constraint just on variables appearing in } S_{t_i} \textit{ and } G_{t_i}\}$($C_{t_i}$ *notably contains constraints on origin and horizon timepoints)*, and $F_{t_i} = G_{t_i}$.

POCL planning cannot be interrupted at any time and come up with an applicable plan. Still we have to guarantee that at the end of the replanning process, there remains enough time to execute the solution plan and meet the goal deadlines. We propose to add a specific flexible timepoint T^{end} to P_{t_i}, that corresponds to the end of the planning process. T^{end} is only constrained to occur between t_i and the end of the horizon. Each time a new timepoint is inserted by the planning process, it is constrained to occur after T^{end}. Thus T^{end}_{ub} decreases as new actions or new temporal constraints are added, and there is not enough time to execute the current plan if $T^{end}_{ub} <$ current time. Note however that T^{end}_{ub} can increase when backtracking.

The strategy is then to plan one step at a time until it results into a dead-end, or a solution is found, or a time limit l is reached. l is defined as $l = T^{end}_{ub} - d$, d being a *slack* duration to save enough time at the end of planning for cycle initialization. l is updated after each planning step. Planning is stopped when l is reached unless the next step corresponds to a backtrack node. In that case, and if the next step increases l, planning is pursued.

If planning is aborted without finding a solution, some goals are rejected and a new attempt is done (Lemai 2004).

Integration and example of scenario

We illustrate the capabilities and the performances of IxTeT with an example of a scenario for a rover with an exploration mission. In such a domain, the quantitative effects and durations can be estimated in advance for planning but are accurately known only at execution time (e.g. the actual compression rate of an image or the actual duration of a navigation task), thus requiring regular updates and look-ahead capabilities to manage unforeseen situations and resource levels. We also illustrate the advantage of using the 3DC+ algorithm in order to produce a more robust plan and compare with a plan without temporal controllability.

IxTeT has been integrated in the decisional level of the LAAS architecture (Alami *et al.* 1998) and

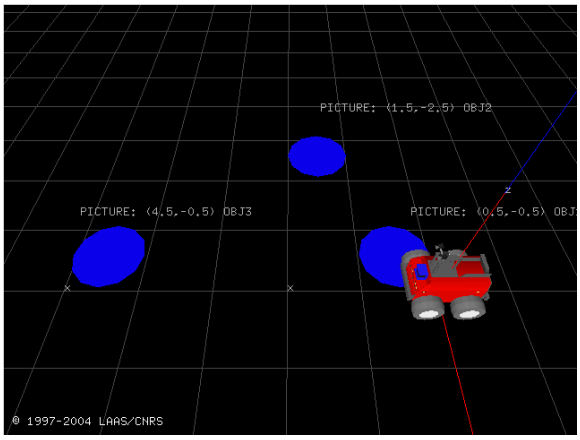

Figure 6: DALA GUI showing the goals of the exploration mission.

used to control an iRobot ATRV (see the first section and Fig. 8). We set up an exploration mission scenario which requires the robot to achieve three types of goals (see Fig. 6): "take pictures of specific science targets" (in locations (0.5,-0.5), (4.5,-0.5), (1.5,-2.5)), "communicate with a ground station during visibility window" ($W_1[117-147]$), and "return to location (0.5,-0.5) before time 500". Dala runs a 3 GHz Pentium IV (1 GB memory) under Linux and is equipped with the following sensors: odometry and a stereo camera pair mounted on a pan&tilt unit (PTU). Five main actions are considered at the mission planning level: take_picture, move_ptu, move (Fig. 2), download_images, communicate. The first three actions are performed by Dala, while the last two are realistically simulated.

There are specific constraints attached to each tasks. The pan&tilt unit must be warmed up ten seconds before it can move. During a move action (of the rover), the camera must be pointed at a specific angle in order to provide the best perception of the environment. Thus the move action and the move_ptu action are mutually exclusive, however the pan&tilt unit can be warmed up during the "end of the move". It allows us to start a move_ptu action before the end of the move that precedes it without "stopping" the move itself. Yet, to do so, we need 3DC+ to correctly produce and execute this plan. Without this, IxTeT produce a plan which may shorten the duration of the move to its lower limit and we will most likely get a temporal failure. You can see on Fig.7 that in the top plan the end of two move action is overlapped by a move_ptu action. Unfortunately at this stage, the IxTeT Plan Viewer used to produce these screen dumps does not show the wait introduced by the 3DC+ algorithm. The plan on the bottom part of the picture has been produced by over constraining the move_ptu action to take place strictly after the move action. This plan is clearly safe but less efficient and flexible than the previous one.

The plan execution is controlled by both executives as

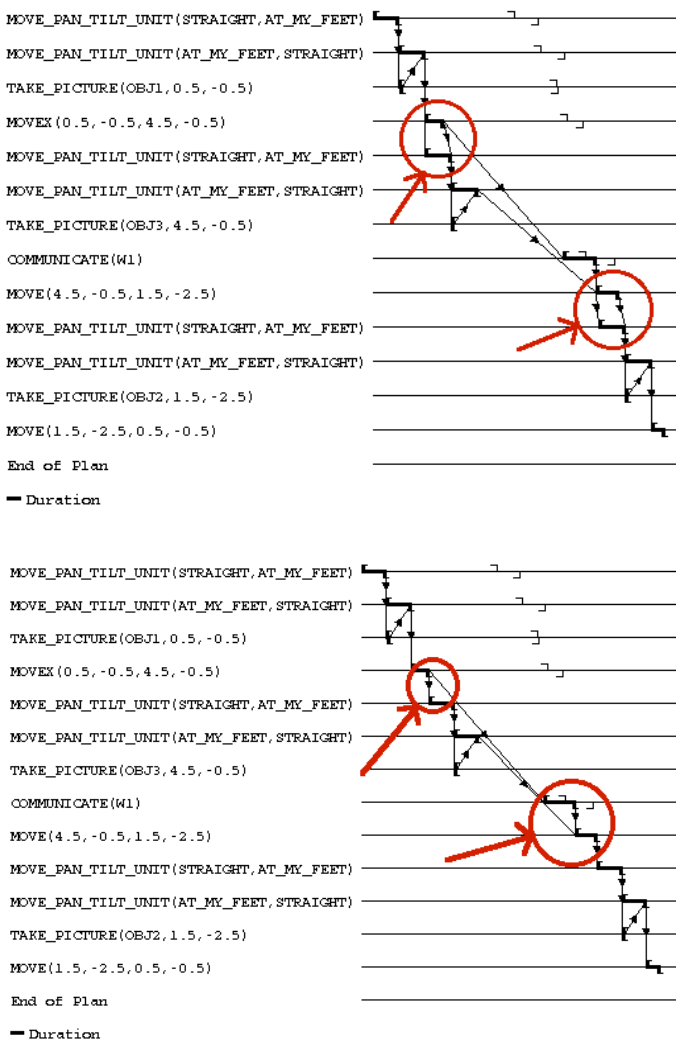

Figure 7: Initial plan produced with the use of 3DC+ (top) and without (bottom) (note the flexibility left, the dependencies and the parallelism).

follows. IxTeT decides when to start or stop an action in the plan and handles plan adaptations. OpenPRS expands the action into commands to the functional level[6], monitors its execution and can recover from specific failures. It finally reports to IxTeT upon the action completion.

This mission (the corresponding initial plan with 3DC+ is shown in Fig. 7) has been executed by Dala under IxTeT control (with $\mu = 1s$ and total cycle duration= $2s$). The initial plan with 3DC+ was produced in $7.1s$, and the plan without 3DC+ was produced in $4s$. Each resulting run is different.

Figure 9 shows the duration of each phase of the cy-

[6]For the download_images and communicate actions, specific procedures simulate the visibility windows and the gradual download of images.

Figure 8: The robot Dala.

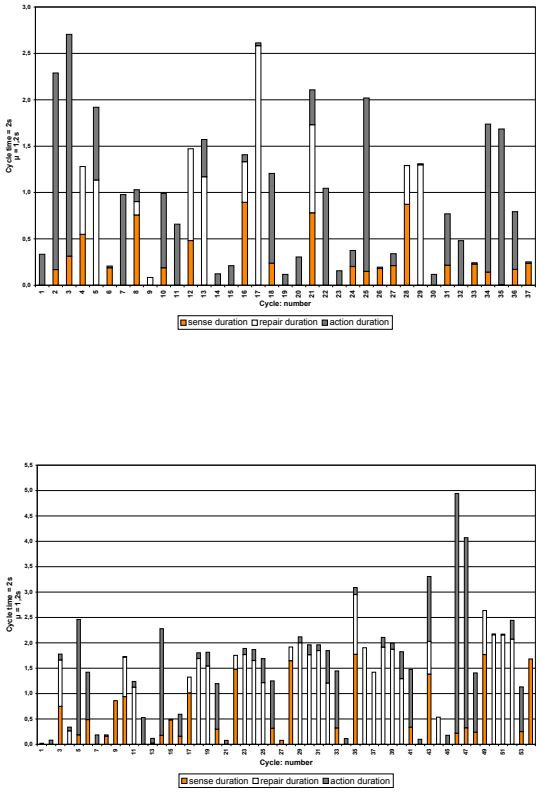

Figure 9: Cycle duration of plans with 3DC+ (top) and without 3DC+ (bottom).

cle for two different runs (one with 3DC+ and another without). The two runs are different because the real execution lead to more frequent failures of the move action in the second run. Yet, we see that using 3DC+ during execution does not increase execution time too much.

Discussion and Prospectives

If one looks at the current state of the art, few high level planning systems have been integrated onboard real robots while running complex navigation software. Many architectures (such as Claraty (Estlin *et al.* 2002)) provide a "decisional" level for such components , but little has been done as far as deploying them entirely on real systems. The main reason is probably that despite the availability of good planning systems, few of them integrate the proper plan repair and replanning mechanisms. Still, the ROGUE system (Haigh & Veloso 1998), for instance, performs planning for asynchronous goals and execution monitoring enhanced with learning capabilities. In (Beetz 2000), the authors propose a different approach where the plans themselves specify the adaptation processes as subplans. In any case, very few approaches explicitly handle time and address the issue of temporal execution. The CASPER system (Chien *et al.* 2000) (part of Claraty) performs continuous planning interleaved with execution. State and temporal data are regularly updated and potential future conflicts are incrementally resolved using iterative repair techniques. However this approach does not handle conflicts which appear within the replanning time interval. Other approaches such as IDEA (Finzi, Ingrand, & Muscettola 2004) are more radical and provide an architecture which seamlessly integrates temporal planning and execution control: each component can be seen as an agent running a reactive planner, and sharing with the others parts of a global temporal model specifying the "behavior" as well as the communication between agents.

We have presented in this paper the IxTeT system which combines a temporal lifted POCL planner with a temporal executive to integrate deliberative planning, execution monitoring and replanning while respecting real-time constraints. This approach cannot account for all the possible execution failures in all their generality. Nevertheless, in many situations where some temporal and resource flexibility has been left,one can expect the presented repair techniques to greatly improve the overall performance of the system by:

- reducing the number of complete replannings,
- improving the system reactivity to unexpected events,
- taking into account new goals on the fly,
- managing the changes in the resources capacity,
- managing the uncertainty in the model description (actions duration, resources consumption/production).

Moreover, by implementing 3DC+ in IxTeT, we have a better handling of temporal controllability, and pro-

duce plans which are more robust at execution time, without a major degradation in performance.

We have conducted a number of field experiments. Although preliminary, the current results are quite promising. First, we show that planning with time and resource combined with execution control, plan repair and replanning can be used on real world problem. Second it shows that such an approach can be deployed on current hardware along with the "state of the art" navigation software (stereo vision, terrain mapping, path planning, visual odometry, etc).

Yet, IxTeT effectiveness can be increased by improving replanning strategies (rejected goals selection, state update requests).

Despite the obvious application of systems such as IxTeT to exploration probes and rovers, one can easily see the possibilities it opens for service robotics (with the added value of human robot interactions and problem joint resolutions) and fields robotics, where planning and execution control problems are also present.

References

Ai-Chang, M.; Bresina, J.; Charest, L.; Jónsson, A.; Hsu, J.; Kanefsky, B.; Maldague, P.; Morris, P.; Rajan, K.; and Yglesias, J. 2003. Mapgen: Mixed initiative planning and scheduling for the mars 03 mer mission. In *Proceedings of iSAIRAS*.

Alami, R.; Chatila, R.; Fleury, S.; Ghallab, M.; and Ingrand, F. 1998. An architecture for autonomy. *International Journal of Robotics Research, Special Issue on Integrated Architectures for Robot Control and Programming* 17(4):315–337.

Beetz, M. 2000. Runtime plan adaptation in structured reactive controllers. In *Proceedings of the Fourth ICAA*.

Chien, S.; Knight, R.; Stechert, A.; Sherwood, R.; and Rabideau, G. 2000. Using iterative repair to improve the responsiveness of planning and scheduling. In *AAAI*.

Estlin, T.; Fisher, F.; Gaines, D.; Chouinard, C.; Schaffer, S.; and Nesnas, I. 2002. Continuous Planning and Execution for an Autonomous Rover. In *Proc. Third International NASA Workshop on Planning and Scheduling for Space*.

Finzi, A.; Ingrand, F.; and Muscettola, N. 2004. Model-based executive control through reactive planning for autonomous rovers. In *IROS 2004 (IEEE/RSJ International Conference on Intelligent Robots and Systems)*.

Garcia, F., and Laborie, P. 1995. Hierarchisation of the search space in temporal planning. In *EWP*.

Goldberg, S.; Maimone, M.; and Matthies, L. 2002. Stereo vision and rover navigation software for planetary exploration. In *Proc. IEEE Aerospace Conference*.

Haigh, K. Z., and Veloso, M. M. 1998. Planning, execution and learning in a robotic agent. In *AIPS*.

Knoblock, C. 1994. Automatically generating abstractions for planning. *Artificial Intelligence* 68.

Laborie, P., and Ghallab, M. 1995. Planning with sharable resource constraints. In *IJCAI*.

Lacroix, S.; Mallet, A.; Bonnafous, D.; Bauzil, G.; Fleury, S.; Herrb, M.; and Chatila, R. 2003. Autonomous rover navigation on unknown terrains, functions and integration. *IJRR*.

Lemai, S., and Ingrand, F. 2004. Interleaving temporal planning and execution in robotics domains. In *AAAI 2004, July 25-29*.

Lemai, S. 2004. *IxTeT-eXeC: planning, plan repair and execution control with time and resource management*. Ph.D. Dissertation, LAAS-CNRS and Institut National Polytechnique de Toulouse, France.

Py, F., and Ingrand, F. 2004. Dependable execution control for autonomous robots. In *IROS 2004 (IEEE/RSJ International Conference on Intelligent Robots and Systems)*.

Rajan, K. 2004. Invited talk: Mapgen. In *IWPSS 2004, 4th International Workshop on Planning and Scheduling for Space, June 23 - 25*.

Trinquart, R., and Ghallab, M. 2001. An extended functional representation in temporal planning : towards continuous change. In *ECP*.

Vidal, T.; Morris, P.; and Muscettola, N. 2001. Dynamic Control of Plans With Temporal Uncertainty. In *IJCAI*, 494–502.